WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Wednesday, June 30, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L30	20001223	8
☐	L29	((data adj4 exchange) and XML).ab.	41
☐	L28	20001223	203
☐	L27	(data adj4 exchange) and XML	1467
☐	L26	20001223	3436
☐	L25	(EDI or (data adj4 exchange)).ti.	4459
☐	L24	L23 and XML	0
☐	L23	((machine-to-machine) and (talk or communication or process)).ab.	13
☐	L22	((machine-to-machine) and (talk or communication or process)).ti.	3
☐	L21	L19 and (data adj3 structure) and (parse or parsing or parser)	0
☐	L20	L19 and XML	1
☐	L19	l16 or l17	72
☐	L18	20001223	1
☐	L17	20001223	19
☐	L16	20001223	60
☐	L15	((business-to-business or e-commerce or (electronic adj3 commerce)) and (automated or automating or automation) and (communication or process or processes)).ab.	113
☐	L14	((business-to-business or e-commerce or (electronic adj3 commerce)) and (automated or automating or automation) and (communication or process or processes)).ti.	4
☐	L13	((business-to-business or e-commerce or (electronic adj3 commerce)) and (automated or automating or automation)).ti.	34
☐	L12	(business-to-business or e-commerce or (electronic adj3 commerce)) and (data adj3 structure) and (parse or parsing or parser)	949
☐	L11	L10 and l9	0
☐	L10	schema.ti.	594
☐	L9	L8 or l7	223
☐	L8	(el-hady or elhady or el adj hady).xp.	82
☐	L7	(el-hady or elhady or el adj hady).xa.	141
☐	L6	L4 and ((detect or detection) near8 (change or changing or addition) near8 (data adj3 structure))	0

☐	L5	L4 and ((change or changing or addition) near8 (data adj3 structure))	37
☐	L4	20001223	87
☐	L3	L2 and XLE	1
☐	L2	L1 and XML	533
☐	L1	(business-to-business or e-commerce or (electronic adj3 commerce)) and (data adj3 structure) and (parse or parsing)	904

END OF SEARCH HISTORY

First Hit

☐ Generate Collection

L30: Entry 2 of 8 File: DWPI Jul 29, 2003

DERWENT-ACC-NO: 2003-707695
DERWENT-WEEK: 200367
COPYRIGHT 2004 DERWENT INFORMATION LTD

TITLE: Data importing method for business to business data interchange, involves receiving user input for selecting set of data structure fields for mapping to XML file data associated tags, in order to populate fields with XML file data

Basic Abstract Text (1):
NOVELTY - The user input are received for selecting data structures within a database and for selecting set of fields belonging to selected structures, and for mapping selected fields to tags associated with data within the XML file. A set of commands that is generated, cause the tags mapped fields to be populated with the data in the XML file.

Basic Abstract Text (2):
DETAILED DESCRIPTION - An INDEPENDENT CLAIM is also included for computer readable medium storing instructions for importing data from XML file to target repository.

Basic Abstract Text (3):
USE - For importing data from extended markup language (XML) file to target repository in business to business data interchange especially for hospitals.

Basic Abstract Text (4):
ADVANTAGE - Allows business to exchange data efficiently without designing special tools for each data exchange format.

Basic Abstract Text (5):
DESCRIPTION OF DRAWING(S) - The figure shows a screen associating selected fields of a selected tables with text, XML tags, sequences or foreign key columns.

PF Application Date (1):
19990804

PF Application Date (2):
20000414

?

First Hit

☐ Generate Collection

L30: Entry 3 of 8 File: DWPI Feb 11, 2003

DERWENT-ACC-NO: 2003-446756
DERWENT-WEEK: 200360
COPYRIGHT 2004 DERWENT INFORMATION LTD

TITLE: Document index creation method in relational database management system, involves defining set of functions in extensible mark-up language extender, to define indices for documents in respective columns of table

Basic Abstract Text (1):
NOVELTY - A set of functions is defined in an extensible mark-up language (XML) extender (100) for processing the XML documents stored in columns of a table. Several indices for the documents are defined on respective columns of the table, using the set of functions.

Basic Abstract Text (5):
USE - For creating indices for extensible mark-up language documents, in relational database management system used for applications such as electronic data interchange for banking exchange, music, chemistry, channel definition format.

Basic Abstract Text (8):
XML extender 100

PF Application Date (1):
19981008

PF Application Date (2):
19990602

First Hit

☐ Generate Collection

L30: Entry 4 of 8 File: DWPI Mar 12, 2002

DERWENT-ACC-NO: 2002-324647
DERWENT-WEEK: 200236
COPYRIGHT 2004 DERWENT INFORMATION LTD

TITLE: Online source data collection and standardization method for data exchange system, involves standardizing source data using XML and distributing standardized data regardless of end user capability

Basic Abstract Text (1):
NOVELTY - Source data that are generated using WIDL generator are collected. The collected data are standardized using extensive mark-up language (XML) and stored. A server program performs automatic conversion and distribution of the standardized data to an end user regardless of the capabilities of end user's system, storage and conversion.

Basic Abstract Text (3):
USE - For data exchange system.

PF Application Date (1):
20000825

First Hit

☐ Generate Collection

L30: Entry 8 of 8 File: DWPI Apr 6, 2001

DERWENT-ACC-NO: 2001-339698
DERWENT-WEEK: 200136
COPYRIGHT 2004 DERWENT INFORMATION LTD

TITLE: Data exchange processing method involves including data searched from database using command character row to structurizing document, based on information read from file

Basic Abstract Text (5):
USE - For data exchange processing of structurizing document produced by extensible markup language (XML), standard generalized markup language (SGML) and hypertext markup language (HTML).

Basic Abstract Text (7):
DESCRIPTION OF DRAWING(S) - The figure shows the entire block diagram of data exchange processing system. (Drawing includes non-English language text).

PF Application Date (1):
19990924

First Hit

☐ Generate Collection

L30: Entry 6 of 8 File: DWPI Jul 25, 2001

DERWENT-ACC-NO: 2001-571406
DERWENT-WEEK: 200165
COPYRIGHT 2004 DERWENT INFORMATION LTD

TITLE: Exchange method and device of XML data information

Basic Abstract Text (1):
NOVELTY - A method and device of XML date information exchange based on ATM cell is disclosed. An XML information package is composed of header and service data. Said header contains at least sourc address ID and destination address ID. The XML data information exchange is implemented by creating the map between said IDs and the route informatino VPI/VCI contained in the header of ATM cell structure. Its advantages are high exchange speed and capacity.

PF Application Date (1):
20000523

No image

First Hit Fwd Refs

☐ Generate Collection

L5: Entry 17 of 37 File: USPT May 27, 2003

DOCUMENT-IDENTIFIER: US 6571282 B1
TITLE: Block-based communication in a communication services patterns environment

Application Filing Date (1):
19990831

Parent Case Text (2):
This application is related to U.S. patent applications entitled A SYSTEM, METHOD AND ARTICLE OF MANUFACTURE FOR A DEVELOPMENT ARCHITECTURE FRAMEWORK, Ser. No. 09/387,747, and A SYSTEM, METHOD AND ARTICLE OF MANUFACTURE FOR MAINTENANCE AND ADMINISTRATION IN AN E-COMMERCE APPLICATION FRAMEWORK, Ser. No. 09/387,318, both of which are filed concurrently herewith and which are incorporated by reference in their entirety.

Drawing Description Text (95):
FIG. 93 depicts conversations with a "null" data structure;

Drawing Description Text (96):
FIG. 94 depicts conversations with a non-"null" data structure;

Drawing Description Text (121):
FIG. 119 illustrates the transmitting of all data in a Data Structure from a client to a server and visa-versa;

Detailed Description Text (4):
OOP is a process of developing computer software using objects, including the steps of analyzing the problem, designing the system, and constructing the program. An object is a software package that contains both data and a collection of related structures and procedures. Since it contains both data and a collection of structures and procedures, it can be visualized as a self-sufficient component that does not require other additional structures, procedures or data to perform its specific task. OOP, therefore, views a computer program as a collection of largely autonomous components, called objects, each of which is responsible for a specific task. This concept of packaging data, structures, and procedures together in one component or module is called encapsulation.

Detailed Description Text (19):
Application frameworks reduce the total amount of code that a programmer has to write from scratch. However, because the framework is really a generic application that displays windows, supports copy and paste, and so on, the programmer can also relinquish control to a greater degree than event loop programs permit. The framework code takes care of almost all event handling and flow of control, and the programmer's code is called only when the framework needs it (e.g., to create or manipulate a proprietary data structure).

Detailed Description Text (166):
Netcentric Computing Top 10 Points Netcentric computing represents an evolution--it builds on and extends, rather than replaces, client/server. Netcentric computing has a greater impact on the entire business enterprise, hence greater opportunity and risk. Definitions of Netcentric may vary. One is about reach and content.

Netcentric is not just electronic commerce; it can impact enterprises internally as well. You can begin identifying Netcentric opportunities for clients today. There are three basic types of Netcentric applications: advertise; inquiry; and fully interactive. One can underestimate the impact of Netcentric on infrastructure requirements. Build today's client/server engagements with flexibility to extend to Netcentric.

Detailed Description Text (240):
Client/server technologies introduced new navigation metaphors. A method for allowing a user to navigate within an application is to list available functions or information by means of a menu bar with associated pull-down menus or context-sensitive pop-up menus. This method conserves screen real-estate by hiding functions and options within menus, but for this very reason can be more difficult for first time or infrequent users. This point is important when implementing electronic commerce solutions where the target customer may use the application only once or very infrequently (e.g., purchasing auto insurance).

Detailed Description Text (269):
Microsoft's Internet Explorer 4.0 supports a W3C "Working Draft" DOM specification that uses the CSS standard for layout control and Web document object manipulation. In contrast, Netscape's implementation of DHTML in Communicator 4.0 uses a proprietary "Dynamic Layers" tag, which assigns multiple layers to a page within which objects are manipulated. As a result, Web pages authored using either version of DHTML may not be viewed properly using the other's browser. XML: X marks the spot

Detailed Description Text (270):
HTML 4.0 and Dynamic HTML have given Web authors more control over the ways in which a Web page is displayed. But they have done little to address a growing problem in the developer community: how to access and manage data in Web documents so as to gain more control over document structure. To this end, leading Internet developers devised Extensible Markup Language (XML), a watered-down version of SGML that reduces its complexity while maintaining its flexibility. Like SGML, XML is a meta-language that allows authors to create their own customized tags to identify different types of data on their Web pages. In addition to improving document structure, these tags will make it possible to more effectively index and search for information in databases and on the Web.

Detailed Description Text (271):
XML documents consist of two parts. The first is the document itself, which contains XML tags for identifying data elements and resembles an HTML document. The second part is a DTD that defines the document structure by explaining what the tags mean and how they should be interpreted. In order to view XML documents, Web browsers and search engines will need special XML processors called "parsers." Currently, Microsoft's Internet Explorer 4.0 contains two XML parsers: a high-performance parser written in C++ and another one written in Java.

Detailed Description Text (272):
A number of vendors plan to use XML as the underlying language for new Web standards and applications. Microsoft uses XML for its Channel Definition Format, a Web-based "push" content delivery system included in Internet Explorer 4.0. Netscape will use XML in its Meta Content Framework to describe and store metadata, or collections of information, in forthcoming versions of Communicator. XML is currently playing an important role the realm of electronic commerce via the Open Financial Exchange, an application developed by Microsoft, Intuit, and CheckFree for conducting electronic financial transactions. Similarly, HL7, a healthcare information systems standards organization, is using XML to support electronic data interchange EDI of clinical, financial, and administrative information (http://www.mcis.duke.edu/standards/HL7/sigs/sgml/index.html).

Detailed Description Text (279):
SMIL is based on XML and does not represent a specific media format. Instead, SMIL defines the tags that link different media typestogether. The language enables Web authors to sort multimedia content into separate audio, video, text, and image files and streams which are sent to a user's browser. The SMIL tags then specify the "schedule" for displaying those components by determining whether they should be played together or sequentially. This enables elaborate multimedia presentations to be created out of smaller, less bandwidth-consuming components.

Detailed Description Text (669):
Web server vendors including Microsoft, Netscape and OpenMarket are putting plans in place to add EDI transmission capabilities into their Web server products. OpenMarket Inc. is working with Sterling and Premenos to integrate their EDI management software with OpenMarkets OMTransact electronic commerce server software. Netscape is working with GEIS in creating Actra Business Systems to integrate EDI services with Netscape server products.

Detailed Description Text (692):
The need for Encryption Services is particularly strong where electronic commerce solutions that involve exchanging sensitive or financial data are to be deployed over public networks such as the Internet. Cryptography can be used to achieve secure communications, even when the transmission media (for example, the Internet) is untrustworthy. Encryption Services can also be used to encrypt data to be stored (e.g., sensitive product information on a sales person's laptop) to decrease the chance of information theft.

Detailed Description Text (693):
There are complex legal issues surrounding the use of encrypting in an international environment. The US government restricts what can be exported (in terms of encryption technology), and the French government defines encryption technology as a "weapon of war" with appropriate legal and regulatory restrictions. This is a key issue in international e-commerce today.

Detailed Description Text (705):
Authentication for accessing resources across an Internet or intranet is not as simple and is a rapidly evolving area. When building e-commerce Web sites there may be a need to restrict access to areas of information and functionality to known customers or trading partners. More granular authentication is required where sensitive individual customer account information must be protected from other customers.

Detailed Description Text (895):
In the Netcentric environment, application security becomes a more critical component primarily because there are more types of users (e.g., employees, customers) and additional types of transactions (e.g., e-commerce, help-desks). In traditional client/server environments most users are employees of the company. In Netcentric environments there are typically also external users (e.g., vendors, registered users) and the general public. Usually, different types of users have different application security requirements limiting what data they can see and what functions they can execute. Also, new types of transactions such as verifying credit when doing e-commerce transactions also require additional application security services.

Detailed Description Text (1169):
The next-generation Web--in its Internet, intranet, and extranet incarnations--must be able to deal with the complex requirements of multi-step business-to-business and consumer-to-business transactions. To do this, the Web must evolve into a full-blown client/server medium that can run your line-of-business applications (i.e., a delivery vehicle for business transaction processing) . . . To move to the next step, the Web needs distributed objects.

Detailed Description Text (1914):
Stream-based communication is a very effective pattern for relaying data, data structures, and meta-data. Meta-data is information about the data like data structure, data types, etc. using a shared, generic format. How can the message format be shared between systems so as to create the most straightforward and best performing stream-based mechanism?

Detailed Description Text (1915):
Often, it is determined that a stream-based communication mechanism should be used to transport information between systems. Stream-based communication is a pattern where information is transported from one system to another using a simple stream and a shared format to relay the data structure and meta-data information.

Detailed Description Text (1918):
On many projects, the following factors influence the details of communicating using a stream. High performance--System performance is always a factor, but sometimes it is one of the most important factors in a system. Short development time--The system must be operational in the shortest possible timeframe. Stable information characteristics--In some solutions, the data and the structure of the data are stable and unlikely to change.

Detailed Description Text (1921):
Fixed format contracts are maps that contain the meta-data information such as the data structure, data separators, data types, attribute names, etc. They describe how to translate Fixed Format messages onto a stream and off of a stream.

Detailed Description Text (1922):
FIG. 67 illustrates an example of a Fixed Format message 6700 associated with the fixed format stream patterns. The location and size of each attribute in the message is fixed and known at design time. In the example below, it is known that the command will be in bytes 1-9, the first name will be in bytes 10-29, the last name in bytes 29-49, etc. This information (meta-data) is used by the Fixed Format contracts to convert Fixed Format messages from data structures to raw data and back again.

Detailed Description Text (1923):
FIG. 68 depicts the complete Fixed Format Stream pattern associated with the fixed format stream patterns. A data structure on System A 6800 is translated to a Fixed Format message (raw data) using a Fixed Format contract. The message is put in the stream 6802 and sent to System B 6804. System B 6804 receives the Fixed Format Message (raw data) and uses its Fixed Format contract to recreate the data structure. The same process works in reverse when System B 6804 responds to the message request.

Detailed Description Text (1924):
Benefits Performance. Because there is no time spent on look-ups or dynamic translation of the message, performance is better than with other variations of Stream-Based Communication. Small Message Size. Each Fixed Format message contains only data to be sent to the other system. These messages contain no meta-data and are smaller than those in Self-Describing Streams. Simplicity. Translating and parsing information onto and off of the stream is straightforward and easier than with the other variations of Stream-Based Communication. The behaviors for the Fixed Format Streaming are contained in the fixed format contracts on the interfaces of the sending and receiving systems and thus easy to find. Object Friendly. This pattern is very straightforward to implement in object based systems. The objects contain the fixed format contracts and manage the translation and parsing onto the stream. These objects can access their own private behaviors which makes the interface much simpler.

Detailed Description Text (1926):
In non-object systems, define a fixed format contract on the parsing interface module of the sending system. The interfacing module on the sending system can use the contract as a map for how to translate and write the data onto the stream. Define a corresponding fixed format contract on the interface modules of the receiving system. The interface module on the receiving system can use the contract to read and translate the data off of the stream.

Detailed Description Text (1927):
In object-based systems, make each object responsible for its own fixed format contract. Using this contract, each object is able to retrieve and parse its attribute values onto a stream as strings (streamOn) and each object class should be able to parse attributes off of a stream and put them into a new instance of an object (streamOff). Also, it is good practice to include the version of the format within the stream so that concurrent format versions can be accommodated in the design.

Detailed Description Text (1929):
FIG. 70 illustrates a Customer object 7000 in an object-based system 7002 streaming itself into a stream 7004, the stream being sent to a non-object system 7006, this stream being read and the data inserted into a relational database 1008. 1. The CustomerObject with attributes name, sex, and age has a method "streamOn: aStream." It is invoked with an empty stream as the argument `aStream`. The CustomerObject "streamOn:" method goes through each of the object's attributes and parses each values as a string onto the stream. The fixed format contract here is embodied in the order that this method parses the attributes onto the stream. A pseudo-code example in Java is the following: Note--Assume that "asString( )" converts the receiver to a string and that "padWithSpaces( )"pads the string with spaces and makes the string the length specified. /** Stream my attribute values on aStream **/ public void streamOn (OutputStream aStream) { aStream.write(this.getName ( ).asString( ).padWithSpaces(10)); aStream.write(this.getSex( ).asString ( ).padWithSpaces(7)); aStream.write(this.getAge( ).asString( ).padWithSpaces (3)); } 2. The stream is then put into a message communication mechanism like MQSeries or MessageQ and sent to the non-object system. 3. Once at the non-object system, interface code reads through the stream, parses the values off of the stream, converts them to the appropriate types if required, and puts them in a copybook with the appropriate structure. In this example, the fixed format contract is embodied in the structure and type of the WS-SHARED-FORMAT-CUSTOMER working-storage copybook. Refer to the pseudo-COBOL example below.

Detailed Description Text (1931):
Collaborations Stream-based Communication. This is the parent pattern to Fixed Format Stream. In this pattern, information is transmitted using a simple stream and a shared, generic format. The Fixed Format Stream is a more specific implementation of Stream-Based Communication. Structure Based Communication--This pattern uses a Fixed Format Stream to transmit data structure between systems. It is often used to obtain data from a Server for display in a Client UI. Bridge (from the Gamma book Design Patterns) describes a way to de-couple an abstraction from its implementation so that the two can vary independently. The Bridge pattern is often used to define collaborations between a business object and a format object while decoupling the business object from its specific stream format. Abstract Factory (from the Gamma book Design Patterns) is a pattern for creating families of related classes. This could be used with the Bridge pattern to retrieve the format dynamically based on non-static information.

Detailed Description Text (2025):
FIG. 90 illustrates a flowchart for a method 9000 for communicating a null value. A query is first communicated in operation 9002 from a first system to a second system to determine whether a data structure is a null value. Next, in operation 9004, a response to the query is received from the second system indicating whether

the data structure is a null value. A request for the data structure is sent from the first system to the second system in operation 9006 only if the response indicates that the data structure is not a null value. Subsequently, the data structure is received from the second system in operation 9008.

Detailed Description Text (2026):
As one option, the response may be a Boolean indication. As another option, the response may be determined based on an attribute of the data structure. As a further option, the data structure may represent a set of a plurality of values. Also, the first system may, optionally, be a client and the second system is a server.

Detailed Description Text (2035):
The extra attribute on the structure then determines whether or not the structure represents a "null" value. The structure can be queried to determine whether or not it represents a "null" value. FIG. 93 depicts conversations 9300 with a "null" data structure 9302. FIG. 94 depicts conversations 9400 with a non-"null" data structure 9402.

Detailed Description Text (2039):
In the code that prepares the data to be sent over the ORB, a check of the data is made and the structure is populated appropriately. If it is null, the isNull flag is set, otherwise it is cleared. Refer to the following code example:

Detailed Description Text (2118):
Stream-based communication is a very effective pattern for relaying data, data structures, and meta-data. Meta-data is information about the data, such as data structure, data types, etc. using a shared, generic format. How can the message format be shared between systems so as to create the most flexible stream-based communication mechanism?

Detailed Description Text (2122):
Many additional factors influence the detailed design of this communication mechanism. Some systems support volatile and constantly changing object models, data models and data structures. In these systems, flexible, de-coupled communication is extremely important.

Detailed Description Text (2123):
In a constantly changing system, a statically defined "shared format" doesn't work very well. Every change to the object model, data model of data structure causes a reimplementation of the "shared format." Each reimplementation results in a redesign, recompile, and retest of the changed code

Detailed Description Text (2133):
Benefits Greater Flexibility. Because the information about the structure of the data has been parameterised and stored as additional data within the message, changes to the data structure would have no effect on this interface mechanism. This means the interface mechanisms will not need to be re-designed/re-built/re-tested/re-deployed, etc for each change in meta-data. Interfacing systems are better de-coupled. Because the message format is embedded in the actual stream, this format does not need to be stored or kept in synch across different systems. It can be "discovered" at run-time when the interface is invoked.

Detailed Description Text (2134):
For object-based systems, the implementation is quite straightforward. Simply make each object responsible for implementing streaming behaviors based on the format and message language. Each object should know how to get and parse its attribute values onto a stream as string values (streamOn) and each object class should know how to parse attributes off of a stream and put these values into a new instance of the object (streamOff).

Detailed Description Text (2136):
FIG. 110 illustrates a Customer object 11000 in an object-based system 11002 streaming itself into a stream 11004, the stream 11004 being sent to a non-object system 11006, this stream 11004 being read and the data inserted into a relational database 11008. The steps illustrated in FIG. 110 will now be set forth. 1. The CustomerObject with attributes name, sex, and age has a method "streamOn: aStream." It is invoked with an empty stream as the argument `aStream`. The CustomerObject "streamOn:" method goes through each of the object's attributes and parses each value as a string onto the stream. In the Java pseudo-code below, the message language defines the format of the header, the format of the attribute descriptors, and the delimiter used in the parsing.

Detailed Description Text (2145):
In order to successfully transmit a formatted message, both the sending and receiving systems must understand the format and structure of the transmitted information. Some communications mediums, however, do not inherently transmit the formatting information with the data. How can information be easily communicated between systems when the communication mechanism does not inherently convey data structure or other meta-data information?

Detailed Description Text (2148):
Other types of middleware, however, do not provide this full range of functionality. Lower level communication protocols like TCP and UDP as well as higher-level protocols like HTTP and telnet do not provide support for sending data structures.

Detailed Description Text (2150):
These communication protocols do not inherently convey meta-data information. `Meta`-data is information about the data. Meta-data could describe the data structure (senders address in bytes 1-10, receivers address in bytes 11-20, data in 21-100, etc.), data types, etc.

Detailed Description Text (2159):
The "shared format" provides the meta-data information needed to interpret the raw data in the buffer. This shared format is like a secret decoder ring for systems sending and receiving messages. The sending system uses the decoder ring to convert objects, structures, etc. into raw data on a stream. The receiving system uses another decoder ring to reconstitute the raw data back into objects or structures. If objects aren't supported, the raw data is converted into a comparable format for use by the receiving system.

Detailed Description Text (2164):
The implementation of Stream Based Communication is very straightforward. Simply define interface code on the sending system that creates a stream and parses the data onto this stream using a format shared by the both the sending and receiving systems. On the receiving system, define interface code that reads the stream and, using the same shared format, parses the data off of the stream and into a data structure compatible with the receiving system.

Detailed Description Text (2166):
For object-based systems, make each object responsible for implementing streaming behaviors that use this shared format. Each object should use the format as a map to parse the attribute values onto a stream (streamOn). Conversely, each object class should use the format as a map to parse its attribute values off of a stream and put them into a newly instantiated instance of the object (streamOff).

Detailed Description Text (2168):
FIG. 115 is an illustration of a Customer object 11500 in an object-based system 11502 streaming itself into a stream 11504, the stream 11504 being sent to a non-

object system 11506, this stream 11504 being read and the information is inserted into a relational database 11508. 1. The CustomerObject with attributes name, sex, and age has a method "streamOn: aStream." It is invoked with an empty stream as the argument `aStream`. The CustomerObject "streamOn:" method goes through each of the object's attributes and parses each values as a string onto the stream. The fixed format contract here is embodied in the order that this method parses the attributes onto the stream. A pseudo-code example in Java is the following: Note--Assume that "asString( )" converts the receiver to a string and that "padWithSpaces ( )" pads the string with spaces and makes the string the length specified.

Detailed Description Text (2171):
Collaborations Fixed Format Stream--This child pattern is a specific variation of Stream-Based communication where the messaging format is defined and stored on both the sending and receiving systems. Downloadable Format Stream--This child pattern is a specific variation of Stream-Based communication where the messaging format is stored at a central location and is downloaded by the communicating parties when needed. Self-Describing Stream. This child pattern is a specific variation of Stream-Based communication where the messaging format is parameterised and stored on the stream. A message language is used to read and write the format of the message from the stream. Structure Based Communication--This pattern uses a Fixed Format Stream to transmit data structure between systems. It is often used to obtain data from a Server for display in a Client UI. Bridge (from the Gamma book Design Patterns) describes a way to de-couple an abstraction from its implementation so that the two can vary independently. The Bridge pattern is often used to define collaborations between a business object and a format object while decoupling the business object from its specific stream format. Abstract Factory (from the Gamma book Design Patterns) is a pattern for creating families of related classes. This could be used with the Bridge pattern to retrieve the format dynamically based on non-static information.

Detailed Description Text (2173):
FIG. 116 illustrates a flowchart for a method 11600 for efficiently retrieving data. A total amount of data required for an application executed by a client is determined in operation 11602. In a single call, the total amount of data from a server is requested over a network in operation 11604. All of the data is bundled in operation 11606 into a data structure by the server in response to the single call. In operations 11608 and 11610, the bundled data structure is sent to the client over the network and the data of the data structure is cached on the client. The cached data of the data structure is used as needed during execution of the application on the client in operation 11612.

Detailed Description Text (2174):
The data structure may be bundled on the server by a business object. In addition, the business object may be instantiated by an action of the client. Also, the network may be at least one of a local area network and a wide area network. As a further option, the request may be administered by a proxy component. Further, the data structure may contain no logic.

Detailed Description Text (2184):
FIG. 119 ilustrates the transmitting of all data in a Data Structure 11900 from a client 11902 to a server 11904 and visa-versa. As shown, to maximize the performance on the client, it is best to bundle all the necessary data into a single data structure that can be transmitted as a structure across the network.

Detailed Description Text (2185):
The Client would first determine the sum total of everything it will need from the business object on the Server machine. The Client makes a request for all of this data from the business object. The business object bundles all the data into a data structure and returns it to the client. The Client will cache this data (using the Caching Proxy pattern) on its local client machine and use it as needed.

Detailed Description Text (2192):
Collaborations 8. The Client asks the Customer Proxy for the data associated with the Jimbo Jones object. 9. The Customer Proxy forwards the request across the network to the Customer Component 10. The Jimbo Jones object creates a data structure and populates it with its data. 11. The Data Structure is passed across the network to the Customer Proxy on the Client. 12. The Customer Proxy forwards the data structure containing Jimbo Jones' data to the Client component.

Detailed Description Text (2193):
Participants Client--The "client" for the transaction. This could be a User Interface that displays customer data for viewing by a Customer Service Representative. Network--A LAN or WAN network that connects the Client with the Customer Component. Customer Component--A server component that encapsulates the data for all of the customers in a system. Customer Component Proxy--A proxy to the Customer Component. Any request it receives, it forwards across the network to the Customer Component. Customer Proxy--A proxy to the Jimbo Jones Customer Object Any request it receives, it forwards across the network to the Jimbo Jones Customer Object. ACustomerStructure--A data structure. It contains the data (but no methods) from the Jimbo Jones object. Database--Any relational database. Jimbo Jones Object--An object that represents the Jimbo Jones customer. This object contains Jimbo Jones' data and methods associated customer methods.

Detailed Description Text (2196):
The following snippet of code defines the data structure used to pass customer information. public CustomerStructure(String firstName, String lastName, short streetNumber, String street, String city, String state, String zipCode)

Detailed Description Text (2201):
Collaborations Proxy--This pattern is documented in the book "Design Patterns" by Gamma, Helm, Johnson and Vlissides. The proxy pattern is often used to communicate with server components in a distributed environment. The Proxy pattern is often used to retrieve data structures from a server component. Cached Proxy--This pattern is documented in "The Proxy Design Pattern Revisited" section of the Pattern Languages of Programming Design 2 book. A Cached Proxy caches data locally on the client. Structure Based Communication uses and builds upon this pattern. Globally Addressable Interface--This pattern often works in conjunction with Structure Based Communication. Oftentimes, a Globally Addressable Interface is used to obtain Structures of data for display on a Client. Locally Addressable Interface--This pattern can also be used in conjunction with Structure Based Communication. After establishing a relationship with an LAI, a client may obtain data from the Server object using Structure Based Communication.

Detailed Description Text (2214):
The client, on the other hand, is mostly responsible for supporting user interactions with the system. To be successful, the client must also execute some degree of business logic. While this can vary from implementation to implementation, some categories of logic are invariably located on the client. This includes simple data validations, representing data structures and relationships, error and exception handling, and communications with the server.

Detailed Description Text (2231):
The presentation object then copies its data into a structure representing some business entity (aNetworkltem 12704) and passes it to the activity 12706. The presentation object then triggers the activity to start its processing of the new network item.

Detailed Description Text (2385):
Large systems can be quite complex and require error management integrating disparate components and/or libraries (i.e. DBMS APIs, data structures library,

middleware, etc) How can exception handling logic be written so that little or no changes are required when new exceptions are added to the system?

Detailed Description Text (2434):
A key issue frequently encountered in the development of object-oriented systems is the mapping of objects in memory to data structures in persistent storage. When the persistent storage is an object-oriented database, this mapping is quite straightforward, being largely taken care of by the database management system.

Detailed Description Text (2506):
Therefore, data access should be organized around business entities, rather than transactions. Individual Persistence packages data into cross-functional objects, rather than transaction-specific data structures. Each individual business object, instead of the window or application, manages the retrieval of its data items.

Detailed Description Text (2573):
Optionally, the state class may support data structures of arbitrary shapes. Supporting classes may manipulate the state in a polymorphic fashion. As another option, the state may be further implemented as a class that contains key-value attribute pairs. The state class may also contain a keyed data structure containing attribute names and attribute values. Additionally, the state can also be asked to write data to a stream.

Detailed Description Text (2585):
Implement the State as a Class Containing Member Variables of Basic Data Types 1. Implementation using a "developer-friendly" state class: A state class can contain basic data types (except char*), extended basic data types with value semantics (e.g., currency, String), other state classes, and vectors (not arrays) of all of the above. This does not, however, imply that the class is a flexible (dictionary-like) data structure. These state classes could/should all inherit from a common abstract base class which defines (but does not implement, at least in C++) a serialization protocol (Java is a different story than C++ because everything is serializable almost by default).

Detailed Description Text (2587):
This is an alternative approach to the one listed above. Using this technique the state class would contain a keyed data structure (e.g. a dictionary) containing keys (the attribute name) and values (the attribute value). In cases where you want to copy the state or pass it to another process, the supporting code does not need to know the type of the state object it is working with. State objects can simply be asked to read or write their data to and from a stream or string. While this offers a more dynamic solution, it should be noted that with this solution additional logic would need to be included in the persistent object to insure the validity of the associated state class.

Detailed Description Text (2593):
Object and component based projects designed and built from the ground up will likely have a well thought out component model and architecture where GUI widgets are linked or bound to domain objects. Data access (and retrieval) for these objects is organized around the business entity, rather than a transaction, and so data is packaged into cross-functional objects, rather than transaction-specific data structures. Each business object manages the retrieval of its data items.

Detailed Description Text (2608):
Benefits Legacy Reuse. The overwhelming benefit of piecemeal retrieval is that it enables reuse of legacy systems. Clients generally have a substantial investment in their existing systems and they will be hard pressed to convert them to component based systems built on objects. This pattern allows new systems to reuse existing business logic through their transactions. Performance and Impedance. Objects based on transactions typically bring back only that data which is needed. The

transaction is typically mapped directly to change that the user is trying to make. This improves performance by reducing the number of network calls and only bringing back data that is needed. Individual Persistence organizes data access around business entities rather than transactions. Object Streaming handles the conversion of data from the structures received from transactions into business objects.

Detailed Description Paragraph Table (6):
... DATA DIVISION. FD FILE-STREAM-IN RECORD CONTAINS 20 CHARACTERS ... WORKING-STORAGE SECTION. ***THIS COPYBOOK CONTAINS THE COMMON FORMAT OF THE ***CUSTOMER IN THE DATA STRUCTURE AND DATA TYPES 01 WS-COMMON-FORMAT-CUSTOMER 03 WS-COMMON-FORMAT-NAME PIC X(10). 03 WS-COMMON-FORMAT-SEX PIC X(7). 03 WS-COMMON-FORMAT-AGE PIC 999. ***THIS COPYBOOK IS THIS SYSTEMS VIEW OF A CUSTOMER 01 WS-CUSTOMER 03 WS-NAME PIC X (10). 03 WS-AGE PIC 999. 03 WS-SEX PIC X(10). ... PROCEDURE DIVISION. ... ***OPEN THE FILE STREAM AND PUT THE CONTENTS IN THE ***WS-COMMON-FORMAT-CUSTOMER COPYBOOK. OPEN FILE-STREAM-IN READ FILE-STREAM-IN INTO WS-COMMON-FORMAT- CUSTOMER AT-END CLOSE FILE-STREAM-IN END-READ. ***MOVE THE VALUES INTO FROM THE COMMON FORMAT INTO ***THE WS-CUSTOMER VARIABLES. MOVE WS-COMMON-FORMAT-SEX TO WS-SEX. MOVE WS-COMMON-FORMAT-AGE TO WS-AGE. MOVE WS-COMMON-FORMAT-NAME TO WS-NAME. ... ***CALL A SQL MODULE TO SAVE THIS INFORMATION IN THE ***RELATIONAL DATABASE CALL "SAVE-CUSTOMER-IN-DATABASE" USING WS- CUSTOMER. ... STOP-RUN.

Detailed Description Paragraph Table (13):
/** Stream my attribute values on aStream **/ public void streamOn (OutputStream aStream) { // CREATE THE HEADER aStream.write(`CUSTOMER `); // This is a customer object aStream.write(`003`); // with three attributes aStream.write(`001`); // this is the format version // DESCRIBE EACH ATTRIBUTE aStream.write(Stream.Delimiter); aStream.write(`NAME `); aStream.write(`STG `); aStream.write(`010`); aStream.write (Stream.Delimiter); aStream.write(`SEX `); aStream.write(`STG `); aStream.write (`007`); aStream.write(Stream.Delimiter); aStream.write(`AGE `); aStream.write(`NUM `); aStream.write(`003`); // WRITE OUT THE ATTRIBUTE VALUES AS DATA aStream.write (Stream.Delimiter); aStream.write(this.getName().asString().padWithSpaces(10)); aStream.write(this.getSex().asString().padWithSpaces(7)); aStream.write(this.getAge ().asString().padWithSpaces(3)); } 2. The stream is then put into a message communication mechanism like MQSeries or MessageQ and sent to the non-object system. 3. Once at the non-object system, interface code reads the stream, parses the values off, converts and moves the values into a copybook with the appropriate structure, and saves the information in relational database. A pseudo-COBOL example is listed below. In reality, this interface code would be more dynamic than depicted in this example.

Detailed Description Paragraph Table (14):
. . . DATA DIVISION. FD FILE-STREAM-IN RECORD CONTAINS 100 CHARACTERS . . . WORKING-STORAGE SECTION. 01 WS-FILE-STREAM-IN PIC X(100). 01 WS-SHARED-FORMAT-HEARDER 03 WS-HEADER-OBJECT-TYPE PIC X(10). 03 WS-HEADER-NUM-OF- PIC X(7). ATTRIBUTES 03 WS-HEADER-VERSION-OF- PIC 999. FORMAT 01 WS-SHARED-FORMAT-ATTRIBUTE 03 WS-ATTRIBUTE-NAME PIC X(5) 03 WS-ATTRIBUTE-TYPE PIC X(5). 03 WS-ATTRIBUTE-SIZE PIC 999. 01 TEMP-VARIABLES 03 WS-INDEX PIC 9999. . . . 01 WS-CUSTOMER 03 WS-NAME PIC X(10). 03 WS-SEX PIC X(7). 03 WS-AGE PIC 999. . . . 88 LT-HEADER-SIZE PIX 99 VALUE 20. 88 LT-ATTRIBUTE-DESCRIPTOR-SIZE PIX X(1) VALUE 14. 88 LT-DELIMINATOR PIX X(1) VALUE ".vertline.". 88 LT-STRING PIX X(1) VALUE "STG ". 88 LT-NUMBER PIX X(1) VALUE "NUM ". . . . PROCEDURE DIVISION. . . . *** OPEN THE FILE STREAM AND READ IT INTO THE TEMPORARY *** *** VARIABLE WS-FILE-STREAM-IN *** OPEN FILE-STREAM-IN. READ FILE-STREAM-IN INTO WS-FILE-STREAM-IN AT-END CLOSE FILE-STREAM-IN END-READ. *** MOVE THE HEADER INFORMATION INTO THE HEADER COPYBOOK**** MOVE (WS-FILE-STREAM-IN FROM ZERO TO LT-HEADER-SIZE) TO WS-SHARED-FORMAT-HEADER. ** FIND WHAT BYTE THE DATA STARTS AT AND SET THE INDEX **** MOVE (LT-ATTRIBUTE-DESCRIPTOR-SIZE * WS-HEADER-NUM-OF-ATTRIBTES) TO WS-INDEX. *** PARSE THE APPROPRIATE OBJECT STRUCTURE OFF OF *** *** THE STREAM *** IF WS-HEADER-OBJECT-TYPE EQUALS "CUSTOMER " THEN PERFORM 1000-PARSE-CUSTOMER-STREAM THRU 1000-PARSE-CUSTOMER-STREAM-END. ELSE IF WS-HEADER-OBJECT-TYPE EQUALS "EMPLOYEE " THEN . . . ELSE IF . . . ELSE *** END THE PROGRAM

RUN-STOP. END-IF. 1000-PARSE-CUSTOMER-STREAM. *** READ WHICH VARIABLE IT IS AND POPULATE THE CORRECT *** *** VARIABLES *** IF (WS-FILE-STREAM FROM WS-INDEX TO (WS-INDEX +5)) = "NAME " THEN MOVE WS-INDEX TO START-INDEX. *** FIND THE DELIMINATOR AFTER THE NAME STRING AND *** *** MOVE THE NAME VALUE INTO THE SEX VARIABLE **** PERFORM VARYING WS-INDEX FROM START-INDEX BY 1 UNTIL (WS-FILE-STREAM-IN AT INDEX) = LT- DELIMINATOR END-PERFORM. MOVE (WS-FILE-STREAM FROM START-INDEX TO WS-INDEX) TO WS-SEX. PERFORM 1000-PARSE-CUSTOMER-STREAM THRU 1000-PARSE-CUSTOMER-STREAM-END. ELSE IF (WS-FILE-STREAM FROM WS-INDEX TO (WS-INDEX + 5)) = "SEX " THEN *** FIND THE DELIMINATOR AFTER THE SEX STRING AND MOVE *** *** THE SEX VALUE INTO THE SEX VARIABLE *** MOVE WS-INDEX TO START-INDEX. PERFORM VARYING WS-INDEX FROM START-INDEX BY 1 UNTIL (WS-FILE-STREAM-IN AT WS-INDEX) = LT- DELIMINATOR END-PERFORM MOVE (WS-FILE-STREAM FROM START-INDEX TO WS-INDEX)TO WS-SEX PERFORM 1000-PARSE-CUSTOMER-STREAM THRU 1000-PARSE-CUSTOMER-STREAM-END. ELSE IF (WS-FILE-STREAM FROM WS-INDEX TO (WS-INDEX +5)) = "AGE " THEN *** FIND THE DELIMINATOR AFTER THE AGE STRING AND *** *** MOVE THE AGE VALUE INTO THE AGE VARIABLE **** MOVE INDEX TO START-INDEX. PERFORM VARYING WS-INDEX FROM START-INDEX BY 1 UNTIL (WS-FILE-STREAM-IN AT WS-INDEX) = LT- DELIMINATOR END-PERFORM MOVE (WS-FILE-STREAM FROM START-INDEX TO WS-INDEX)TO WS-AGE PERFORM 1000-PARSE-CUSTOMER-STREAM THRU 1000-PARSE-CUTOMER-STREAM-END. ELSE PERFORM 2000-SAVE-CUSTOMER THRU 2000-SAVE-CUSTOMER-END. END-IF. 1000-PARSE-CUSTOMER-STREAM-EXIT. 2000-SAVE-CUSTOMER. *** CALL A SQL MODULE TO SAVE THIS INFORMATION IN THE *** RELATIONAL DATABASE CALL "SAVE-CUSTOMER-IN-DATABASE" USING WS- CUSTOMER . . . 2000-SAVE-CUSTOMER-END.

Detailed Description Paragraph Table (15):
/**Stream my attribute values on aStream**/ public void streamOn (OutputStream aStream) { aStream.write(this.getName().asString().padwithSpaces(10)); aStream.write(this.getSex().asString().padwithSpaces(7)); aStream.write(this.getAge ().asString().padWithSpaces(3)); } 2. The stream is then put into a message communication mechanism like MQSeries or MessageQ and sent to the non-object system. 3. Once at the non-object system, interface code reads through the stream, parses the values off of the stream, converts them to the appropriate types if required, and puts them in a copybook with the appropriate structure. In this example, the fixed format contract is embodied in the structure and type of the WS-SHARED-FORMAT-CUSTOMER working-storage copybook. Refer to the pseudo-COBOL example below.

Detailed Description Paragraph Table (16):
. . . DATA DIVISION. FD FILE-STREAM-IN RECORD CONTAINS 20 CHARACTERS . . . WORKING-STORAGE SECTION. *** THIS COPYBOOK CONTAINS THE SHARED FORMAT OF THE *** CUSTOMER IN THE DATA STRUCTURE AND DATA TYPES 01 WS-SHARED-FORMAT-CUSTOMER 03 WS-SHARED-FORMAT-NAME PIC X(10). 03 WS-SHARED-FORMAT-SEX PIC X(7). 03 WS-SHARED-FORMAT-AGE PIC 999. *** THIS COPYBOOK IS THIS SYSTEMS VIEW OF A CUSTOMER 01 WS-CUSTOMER 03 WS-NAME PIC X(10). 03 WS-AGE PIC 999. 03 WS-SEX PIC X(10). . . . PROCEDURE DIVISION. . . . *** OPEN THE FILE STREAM AND PUT THE CONTENTS IN THE *** WS-SHARED-FORMAT-CUSTOMER COPYBOOK. OPEN FILE-STREAM-IN READ FILE-STREAM-IN INTO WS-SHARED-FORMAT-CUSTOMER AT-END CLOSE FILE-STREAM-IN END-READ. *** MOVE THE VALUES INTO FROM THE SHARED FORMAT INTO *** THE WS-CUSTOMER VARIABLES. MOVE WS-SHARED-FORMAT-SEX TO WS-SEX. MOVE WS-SHARED-FORMAT-AGE TO WS-AGE. MOVE WS-SHARED-FORMAT-NAME TO WS-NAME. . . . *** CALL A SQL MODULE TO SAVE THIS INFORMATION IN THE *** RELATIONAL DATABASE CALL "SAVE-CUSTOMER-IN-DATABASE" USING WS- CUSTOMER. . . . STOP-RUN.

Detailed Description Paragraph Table (18):
// Customer Component Code here public class CustomerComponent { // Put the data associated with a Customer Object // into a data Structure. This data structure // will be sent across the network to a client. public Customer getCustomer(String aCustomerName) { // Find the Customer in the database . . . // Instantiate the Customer Object Customer aCustomer = new Customer(. . . . . . . // Return a "remote object reference" to the // Jimbo Jones Customer object. return (aCustomer); } }

Detailed Description Paragraph Table (19):

// Customer object code here public class Customer { // Put the data associated with a Customer Object // into a data Structure. This data structure // will be sent across the network to a client. public CustomerStructure getCustomerAsStructure( ) { CustomerStructure aCustomerStructure = new CustomerStructure( ); aCustomerStructure.firstName = this.getFirstName( ); aCustomerStructure.lastName = this.getLastName( ); aCustomerStructure.streetNumber = this.getStreetNumber( ); aCustomerStructure.street this.getStreet( ); aCustomerStructure.city = this.getCity( ); aCustomerStructure.state = this.getState ( ); aCustomerStructure.zipCode = this.getZipCode( ); return aCustomerStructure; } // Getters and Setters for all attributes // (code not shown here) . . . }

Detailed Description Paragraph Table (39):
class StateClass public: virtual void read(Stream inStream) = 0; virtual void write (Stream outStream) = 0; }; class StateData : public StateClass { public: virtual void read(Stream inStream); // implementation reads state data from stream. virtual void write(Stream outStream); // implementation writes state data to stream. Private: long id; char code[8]; string value; }; 2. Implementation using an enhanced kind of the class described in (A) but which also happens to be a flexible data structure (in the sense that the same class can, similar to a dictionary, support data structures of arbitrary shapes). This approach provides more flexibility since some common behavior can be abstracted into a base class. In addition, supporting classes which need to manipulate the state object (in order to retrieve data from the database or pass the value between processes) can do so in a polymorphic fashion. Also, copy constructors and destructors can be used to handle dynamically allocated memory.

First Hit

☐ Generate Collection

L20: Entry 1 of 1　　　　File: DWPI　　　　Nov 13, 2003

DERWENT-ACC-NO: 2003-901833
DERWENT-WEEK: 200382
COPYRIGHT 2004 DERWENT INFORMATION LTD

TITLE: Standardized data transmission provision method involves translating electronic request received from user into standardized XML format, and accordingly establishing connection between user and non-affiliated user platforms

Basic Abstract Text (1):
NOVELTY - The electronic request received from a user is formatted by translating data contained in the request into normalized XML format recognizable by several platforms of non- affiliated users, using an adapter. Several communication channels are established to enable platform used by the user to communicate with the non-affiliated user platforms.

Basic Abstract Text (3):
USE - For enabling standardized transmission of data between user and financial transactions including online banking, automated teller machine transaction, smart card transaction, electronic fund transfers, electronic bill presentment and payment, business-to-business payment, business-to-business banking, automated teller machine communication, consumer-to- business payment, consumer-to-business banking, wireless transaction, consumer-to-consumer transactions, using user authentication information including biometrics, digital certificates, smart cards. Also for credit unions, savings and loan institution, trust companies and exchange bureaus, financial investment firms.

Basic Abstract Text (4):
ADVANTAGE - The transfer of secure information from user to desired institution is enabled effectively and securely by using data transmission provision system that converts received message into standardized XML format and accordingly establishes link between user and non-affiliated user platforms.

PF Application Date (1):
20000525

First Hit

☐ Generate Collection

L22: Entry 2 of 3 File: DWPI Dec 5, 2002

DERWENT-ACC-NO: 2003-266398
DERWENT-WEEK: 200326
COPYRIGHT 2004 DERWENT INFORMATION LTD

TITLE: Machine-to-machine communication system for network enabled devices, has network through which information from one information device is transmitted to the other for processing, and then fed back to the information device